1) Black Box Testing:

1. Functions:

1.1 row_swap(i,j):

x = [2, -1, 1, 8], [11, 4, 2, 0], [9, 8, 5, 9], [12, -7, -1, 6]
y = [-1, 2], [3, 5], [1, 2]
z = [1, -1, 3], [3, 4, 2], [0, 2, 5]

Test 1: In first test, we swap row1 and row3 of matrix x, We observed our result is correct both rows are correctly swapped.
Function call: x.row_swap(1 , 3)
Result : [2, -1, 1, 8], [12, -7, -1, 6], [9, 8, 5, 9], [11, 4, 2, 0]

Test 2: In second test, we try to swap the rows that do not exist in matrix y. So we are expecting an error.
Function call: y.row_swap(1 , 3)
Result: IndexError: Index out of range: a[3]

Test 3: In third test, we use negative numbers in index as we know negative number in index is allowed in python. Bottom row index is -1 and then next is -2. We used matrix z in this test.
Function call: z.row_swap(0,-2)

Result: [3, 4, 2], [1, -1, 3], [0, 2, 5]

1.2 col_swap(i,j):

x = [2, -1, 1, 8], [11, 4, 2, 0], [9, 8, 5, 9], [12, -7, -1, 6]
y = [-1, 2], [3, 5], [1, 2]
z = [1, -1, 3], [3, 4, 2], [0, 2, 5]

Test 1: In first test, we swap col0 and col3 of matrix x, We observed our result is correct both columns are correctly swapped.
Function call: x.col_swap(0 , 3)
Result : [8, -1, 1, 2], [0, 4, 2, 11], [9, 8, 5, 9], [6, -7, -1, 12]

Test 2: In second test, we try to swap the columns that do not exist in matrix y. So we are expecting an error.
Function call: y.col_swap(1 , 3)
Result: IndexError: Index out of range: a[3]

Test 3: In third test, we use negative numbers in index as we know negative number in index is allowed in python. Right most column has index -1 and then next towards left is -2. We used matrix z in this test.
Function call: z.col_swap(0,-2)
Result: [-1, 1, 3], [4, 3, 2], [2, 0, 5]

1.3 det(A):

A = [-1, 1, 3], [4, 3, 2], [2, 0, 5]
B = [2,3,6,7,9,6]

Test 1: In first test, we have determinant of matrix A. The result is compared with the solution given in a book and found determinant value is correct.
Function call: det(A)
Result: -24

Test 2: det() function takes a square matrix as an argument. So in 2nd test, we give a non- square matrix and observed the result. We are expecting an error.
Function call: det(B)
Result: sympy.matrices.common.ShapeError: Det of a non-square matrix

Test 3: In 3rd test, we did not give any argument and we are expecting an error. We found a self explanatory error.
Function call: det()
Result: TypeError: det() takes exactly 1 argument (0 given)

1.4 eigenvals():

K = [2, 0], [1, 3]
A = [-20, 10], [10, -10]
Y = [-1, 2], [3, 5], [1, 2]

Test 1: In this test, we took an example matrix K from a book that eigenvals is given. We observed eigenvalues calculated by eigenvals function. We found the same eigenvalues with their multiplicity. So function is working correctly.

Function call: K.eigenvals()
Result: {2: 1, 3: 1}

Test 2: In 2nd test, we took an example matrix A that given us complex eigenvalues. So we can check if the function is working fine to calculate complex eigenvalues.
Function call: A.eigenvals()
Result: {-15 + 5*sqrt(5): 1, -15 - 5*sqrt(5): 1}

Test 3: In 3rd test, we took an non-square matrix y. We are expecting an error.
Function call: y.eigenvals()
Result: sympy.matrices.common.NonSquareMatrixError

1.5 eigenvects():

B = [3, -2, 4, -2], [5, 3, -3, -2], [5, -2, 2, -2], [5, -2, -3, 3]

Test 1: We took an online example matrix B for that we know eigenvectors. So we used eigenvects() function and find eigenvectors. We observed, we have the same eigenvectors as we have in example.
Function call: B.eigenvects()
Result: [(-2, 1, [Matrix([
[0],
[1],
[1],
[1]])]), (3, 1, [Matrix([
[1],
[1],
[1],
[1]])]), (5, 2, [Matrix([
[1],
[1],
[1],
[0]]), Matrix([
[0],
[-1],
[0],
[1]])])]

Test 2: In 3rd test, we took an non-square matrix y. We are expecting an error.
Function call: y.eigenvects()

Result: sympy.matrices.common.NonSquareMatrixError

1.6 eye(A):

Test 1: eye function return a square identity matrix of given size. In 1st test, we have given 3 x 3 size matrix to test our function. It is a small matrix and easy to find if function is working correctly. We found our function is working correctly.
Function call: eye(3)
Result: [1, 0, 0], [0, 1, 0], [0, 0, 1]

Test 2: In this test, we produce a bigger matrix of size 20, as function is working correctly for small matrix so we are expecting it will be working correctly with bigger size matrix also.
Function call: eye(20)
Result: [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]

Test 3: eye() function works with non-negative number so we test eye() with a negative number and observed its output. We are expecting an error.
Function call: eye(-3)
Result: ValueError: Cannot create a -3 x -3 matrix. Both dimensions must be positive

Test 4: eye() function works with integer number so we test eye() with a non-integer number and observed its output. We are expecting an error.
Function call: eye(3.5)
Result: ValueError: 3.5 is not an integer.

Test 5: In 5th test, we don't give any argument i**n** eye() function to test if we get an empty matrix but we found it returns an error.
Function call: eye()
Result: TypeError: eye() takes at least 2 arguments (1 given).

1.7 cholesky():

This is one of decomposition methods, it is a decomposition of a Hermitian, positive-definite matrix into the product of a lower triangular matrix and its conjugate transpose.

A = [4, 12 ,-16] , [12, 37, -43] , [-16, -43, 98]
B = [25, 15, -5] , [15, 18, 0] , [-5, 0, 11]
Y = [-1, 2], [3, 5], [1, 2]

Test 1: In 1st test, we used matrix A and used cholesky function. Example matrix A took from wikipedia. We compare our result given in Wikipedia to check if function is working correctly. We found function working correctly.
Function call: A.cholesky()
Result: Function call: A.cholesky()
Result: [2, 0, 0], [6, 1, 0], [-8, 5, 3]

Test 2: In 2nd test, we used matrix B and used cholesky function. Example matrix B took from sympy documentation. We compare our result given in documentation to check if function is working correctly. We also confirm results by multiplying a resultant matrix with its transpose.
Function call: B.cholesky()
Result: [5, 0, 0], [3, 3, 0], [-1, 1, 3] and product = [25, 15, -5] , [15, 18, 0] , [-5, 0, 11]

Test 3: In 3rd test, we applied cholesky() function a non-square matrix, we are expecting an error.
Function call: y.cholesky()
Result: sympy.matrices.common.NonSquareMatrixError: Matrix must be square.

2) White Box Testing:

```
def eigenSpace(eigenval):
        "Getting a basis for the eigenspace for a specific eigenvalue"
        B = Mtx - eye(Mtx.rows)*eigenval
        NullSpace = B.nullspace(iszerofunc = my_iszero)

        if len(NullSpace) == 0:
                NullSpace = B.nullspace(iszerofunc = my_iszero, simplify=True)
        if len(NullSpace) == 0:
                raise NotImplementedError("Can't evaluate eigenvector for eigenvalue
%s" % eigenval)
        return NullSpace
```

We first divide the function in boxes

Block 1:

```
B = Mtx - eye(Mtx.rows)*eigenval
```

Block 2:

```
NullSpace = B.nullspace(iszerofunc = my_iszero)
```

Block 3:

```
raise NotImplementedError("Can't evaluate eigenvector for eigenvalue %s" %
eigenval)
```

Block 4:

```
return NullSpace
```